A suggested approach to computer arithmetic for designers of multi-valued logic processors

Full text Pdf (760 KB)

Source **Multiple-Valued Logic** archive
Proceedings of the eighth international symposium on Multiple-valued logic table of contents
Rosemont, Illinois, United States
Pages: 33 - 46
Year of Publication: 1978

Author D. E. Atkins

Sponsors SIGDA: ACM Special Interest Group on Design Automation
IEEE : Institute of Electrical and Electronics Engineers

ABSTRACT

An approach to the topic of computer arithmetic is suggested which may have a particular conceptual, pedagodical, and practical appeal to the designer of multiple-valued logic processors. Computer arithmetic deals with the physical representation of finite sets of numbers and the design, analysis, and implementation of algorithms for mechanizing arithmetic operations on these sets. Finite number representation systems (FNRS) are specified by defining a set of symbols and a mapping from the elements of this symbol set to a subset of the real numbers. A formal definition of a FNRS provides a basis for a set of definitions which in turn provide the framework for the classification of a large set of number systems. Emphasis in this paper is on the following positive, fixed radix systems: unsigned, sign and magnitude, radix complement, and diminished radix complement. We offer an annotated listing of primitive digit vector algorithms for the four common number representation systems with an arbitrary, positive integer, fixed radix. These digit vector algorithms are ones which the designer of multi-valued logic arithmetic processors will need to implement to provide general arithmetic computation.

REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

1 Sandra Pakin, APL 360 Reference Manual, Second Edition, SRA, Chicago, 1972.

2 G. Demars, J. C. Rault, G. Ruggio, Le Langage et Les Systems APL, (in French), Masson et Cie, Paris, 1974.

3 APL 360 User's Manual, IBM Data Processing Division.

4 APL Shared Variables (APLSV) User's Guide, IBM (SH20-1460).

5 APL Language, IBM, (GC26-3847).

6 University of Michigan Computing Center Memo 363, MTS APL User's Guide.

7 Leonard Gilman , Allen J. Rose, APL: An Interactive Approach, John Wiley & Sons, Inc., New York, NY, 1976

8 K. E. Iverson, A Programming Language, Wiley, New York, 1962.

9 H. Katzan, APL Programming and Computer Techniques, van Nostrand Reinhold, 1970.

10 S. Budkowski, D. Atkins, "A unified classification of finitè number systems," Systems Engineering Lab. Report No. 106, ECE Dept., University of Michigan, Ann Arbor.

11 D. Atkins, "The role of redundancy in computer arithmetic," Computer, June 1975, Vol. 8, No. 6, pp. 74-76.

12 D. Matula, "Radix arithmetic: Digital algorithms for computer architecture," Chapter 9 in Applied Computation Theory: Analysis, Design, Modeling, R. Yeh, ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

13 A. Avizienis, "Digital Computer arithmetic: A unified algorithmic specification," Proceedings of the Symposium on Computers and Automata, New York, 1971, Polytechnic Press, New York. Available through Wiley-Interscience.

CITINGS

D. E. Atkins , W. Liu , S. Ong, Overview of an Arithmetic Design System, Proceedings of the eighteenth design automation conference on Design automation, p.314-321, June 29-July 01, 1981, Nashville, Tennessee, United States

INDEX TERMS

Primary Classification:
G. Mathematics of Computing
↪ **G.1** NUMERICAL ANALYSIS
↪ **G.1.0** General
↪ **Subjects:** Computer arithmetic

Additional Classification:
B. Hardware
↪ **B.6** LOGIC DESIGN

D. Software
↪ **D.3** PROGRAMMING LANGUAGES
↪ **D.3.2** Language Classifications
↪ **Subjects:** Applicative (functional) languages

General Terms:
Design

Collaborative Colleagues:
D. E. Atkins: Janis Beltch Baron
D. J. DeWitt
W. Liu
R. M. Lougheed
T. N. Mudge
S. Ong
R. A. Rutenbar
M. S. Schlansker

A SUGGESTED APPROACH TO COMPUTER ARITHMETIC FOR DESIGNERS OF MULTI-VALUED LOGIC PROCESSORS

D. E. Atkins

Program in Computer, Information and Control Engineering
and
Systems Engineering Laboratory
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

An approach to the topic of computer arithmetic is suggested which may have a particular conceptual, pedagodical, and practical appeal to the designer of multiple-valued logic processors. Computer arithmetic deals with the physical representation of finite sets of numbers and the design, analysis, and implementation of algorithms for mechanizing arithmetic operations on these sets. Finite number representation systems (FNRS) are specified by defining a set of symbols and a mapping from the elements of this symbol set to a subset of the real numbers. A formal definition of a FNRS provides a basis for a set of definitions which in turn provide the framework for the classification of a large set of number systems. Emphasis in this paper is on the following positive, fixed radix systems: unsigned, sign and magnitude, radix complement, and diminished radix complement.

We offer an annotated listing of primitive digit vector algorithms for the four common number representation systems with an arbitrary, positive integer, fixed radix. These digit vector algorithms are ones which the designer of multi-valued logic arithmetic processors will need to implement to provide general arithmetic computation.

INTRODUCTION

Although the predominance of two-valued logic has naturally led to the implementation of binary (radix 2) arithmetic in most digital computers, the theory of computer arithmetic deals with a much broader class of possibilities. At a minimum, the designer of processors with multi-valued technology will be interested in higher radix versions of standard radix polynomial systems, and the possibility exists that more novel number systems (residue, signed-digit, rational, etc.) may find practical application in a multi-valued environment.

An ongoing project within our laboratory concerns developing a unified description and classification of finite number representation systems together with a set of primitive building blocks for arithmetic design, so-called "digit-vector algorithms." One of our goals is to present the designer with a wide range of choices and a method for assigning a figure of merit to various number systems with respect to a given implementation environment. For example, the complexity of the SUM digit vector algorithm is lower in a residue number system than in a standard radix polynomial system, however, the reverse is true for the SIGN (sign detection) digit vector algorithm. The act of defining these algorithms which simulate useful abstract arithmetic structures is what we call "arithmetic design"; traditional "logic design" comes into play only after we make the decision to represent the required digits using binary codes. As often noted by Avizienis, failure to make the distinction between "arithmetic design" and "logic design" has long plagued the literature in computer arithmetic.

The goal of this paper is to encourage collaboration between arithmetic design and the implementation of multi-valued logic. We feel that our first step must be to present definitions and notation to facilitate precise communication, and to enhance appreciation of the wide range of theoretical options available to designers. Specifically, in this paper we present definitions relating to the representation of numbers, give formal definitions of several fixed radix systems in common use, and then list a set of arithmetic building blocks, digit vector algorithms (DVAs), for performing arithmetic in these number systems.

The "approach" we are suggesting is that designers of multi-valued logic processors use the DVAs as formal definitions of the functions which they must provide in their particular circuit technology. This approach should, at a minimum, facilitate the comparison of alternate choices of number systems for given constraints, and also help to avoid the pitfalls which sometimes arise when we too quickly generalized from our radix 2 arithmetic experience. Our suggestion is to think broadly, to learn the general case, and to treat binary arithmetic only as the special case it is.

We shall use the programming language APL as a description language. Although APL is sometimes criticized for having awkward control structures and for encouraging obscurity, we feel that it is well suited for the task at hand. Knowledge of only a relatively small subset of the language is required here. References on APL include [1-9].

This work was supported by the National Science Foundation, Division of Mathematical and Computer Science under Grant No. MCS 77-03310.

REPRESENTATION OF NUMBERS

Computer arithmetic deals with the physical representation of finite sets of numbers and the design, analysis, and implementation of algorithms for mechanizing arithmetic operations on these sets. We consider numbers to be abstract entities which are defined theoretically, typically by their properties (axiomatically). Peano's Five Axioms, for example, define the properties of positive integers. In implementing computer arithmetic we assume we are given the algebra of real and complex numbers. Complex numbers (imaginary numbers) may be made to correspond to a unique pair of real numbers and therefore will not be further discussed explicitly. (Alternately we could view real numbers as being embedded in the class of complex numbers and focus on a discussion of complex numbers.) The set of real numbers include, for example, the natural numbers

$$\underline{N} = \{0, 1, 2, 3, \ldots\},$$

the integers

$$\underline{Z} = \{0, \pm 1, \pm 2, \pm 3, \ldots\},$$

the positive integers

$$\underline{P} = \{1; 2, 3, \ldots\},$$

and the rational numbers

$$\underline{Q} = \{x \text{ such that } x = p \div q \text{ and } p \in \underline{Z} \text{ and } q \in \underline{P}\}.$$

Since we are concerned with the physical mechanization of arithmetic we are restricted to representation of finite sets of numbers, i.e., to subsets of the reals. We will be particularly concerned with representation of, and operations on, the set of integers modulo N,

$$\underline{ZN} = \{0, 1, 2, \ldots, (N-1), \text{ for } N \geq 2\}.$$

Rational numbers (fractions) may be treated either as an ordered pair of integers or as "scaled integers."

When we deal with arithmetic, we imply that such sets of numbers are part of an algebraic system or structure consisting of a set and one or more n-ary operations (functions) on the set. A more complete definition also frequently includes relations on the sets and distinguished elements of the set such as 0 and 1. Examples of algebraic systems include:

$\langle \underline{Z}, +, * \rangle$ where + and * are the operations of addition and multiplication on the set of integers;

$\langle \underline{R}, +, * \rangle$, where + and * are addition and multiplication on the set of reals; and

$\langle \underline{ZN}, +N \rangle$, where +N is addition module N.

To repeat then, computer arithmetic deals with the representation of finite sets of numbers which are typically subsets of the sets described above, and with the mechanization of well-known n-ary (usually unary and binary) operations on these finite sets.

In this section we will consider the representation of numbers by symbols which are variously referred to in the literature as 1) symbol strings, 2) n-tuples, 3) code words, or 4) digit vectors. All of these terms offer some expressive advantage; "symbol strings" relates to language and data structure ideas; "n-tuple" is a standard term in discrete mathematics; "code word" conveys the well-known idea of encoding information; and "digit vector" implies a connection with vector notation and vector languages such as APL. We will reserve the right to use all three, however, our preference will be "digit vector."

We will define finite sets of symbols which can be physically represented and also operations on these sets which "simulate" well-known algebraic structures such as mentioned above. This notion of a "simulation" may be described in terms od the formal idea of homomorphism and all of the various specific subvarieties such as an isomorphism.

It is also intuitively useful to think about (finite precision) computer arithmetic as an approximation of arithmetic on the reals. The fact that it is an approximation gives rise to need for numerical analysis. Only finite precision arithmetic is mechanized on a digital computer. This is probably the most important characteristic of computer arithmetic: direct consequences of finitude include overflow, underflow, scaling, and the use of complement representation of negative quantities. In selecting a number representation system, the designer of a computer arithmetic system must keep in mind the architectural realities of time and hardware efficiency, and the numeric reality of providing an adequate approximation of real arithmetic.

Definitions

Finite number representation systems ("number systems") are usually specified by defining a set of symbols, $\underline{A}$, and a mapping from the elements of this symbol set to a subset of the set of real numbers, $F:\underline{A}\ \underline{R}$.

The elements of the symbol set, $\underline{A}$, usually have the form of a finite N-tuple which we shall call a "digit vector." In other words, a digit vector X is an element of the set $\underline{A}$ where

$$\underline{A} = \underline{D1} \times \underline{D2} \times \ldots \times \underline{DN}$$

and $\underline{DI}$ is the set of allowable digit values for the Ith component and x denotes the Cartesian product.

Following the suggestions of Budkowski [10] we now introduce the following definitions:

1) The function $F:\underline{A}\rightarrow\underline{R}$ is a total function if F is defined for all elements of $\underline{A}$ (all digit vectors in $\underline{A}$).

2) The function $F:\underline{A}\rightarrow\underline{R}$ is a partial function if F is not defined for all elements of $\underline{A}$. In this

case the subset of A for which F is defined is denoted AE and is called the set of "legal digit vectors." Note that in this case $AE \subset A$.

3) Since $AE \subseteq A$ is a finite set, the mapping $F:AE \rightarrow R$ is into R, the infinite set of all reals. The subset $RE \subseteq R$ which is the image of AE under F is called the set of representable numbers or the interpretation set.

The elements of a digit set, DI, are integers and will be taken as defined in this presentation. Note that in practice the elements of a digit set might be denoted using other than standard decimal notation. For example, in the hexadecimal system the digit values are usually taken to be elements of the set {0,1, . . ., 9, A, B, C, D, E, F}. For additional generality numerical meanings can be associated with digit symbols by definition of a one-to-one function from symbols to numbers. Formally then,

Definition 1 A finite number representation system (abbreviated FNRS) is a triple

$$FNRS = \langle A, AE, F \rangle$$

where
A is a finite, nonempty set of digit vectors,
$AE \subseteq A$ is the set of all legal digit vectors, and
F is a function which maps AE into R (the set of real numbers).

In this paper we concentrate on representation of integers, i.e., RE will be a finite set of integers. Having developed integer representation, we may treat representation of rational numbers (fractions) by "scaling" integers.

Definition 2 A FNRS is said to be partial if F is a partial function in A, that is if $AE \subset A$.

Definition 3 A FNRS is said to be total if F is a total function in A, that is if $AE = A$.

Definition 4 A FNRS is said to be redundant if for $X \in AE$, F(X) is a many-to-one function. In this case at least one element of the set of representable numbers (RE) has more than one representation.

Definition 5 A FNRS is said to be nonredundant if for $X \in AE$, F(X) is a one-to-one function.

Although an initial reaction may be that a redundant FNRS would be a disadvantage, in mechanizing machine arithmetic redundancy may offer significant advantage. A discussion of this topic is beyond the scope of this paper but may well be of practical significance in the realization of arithmetic using multi-valued logic. For a brief overview of the role of redundancy in computer arithmetic see [11].

Definition 6 A FNRS is weighed if F is defined by the function

$$Q = (\Sigma X[I] \times W[I]) + C$$

where

$Q \in R$,
$N = \rho X$, the length of digit vector X,
X[I] is the Ith element of a digit vector $X \in AE$,
W[I] is the Ith element of a weight vector with $W[I] \in R$,
and C is a constant.

In APL, the above expression can be written

$$Q \leftarrow (+/X \times W) + C$$

where $\rho X = \rho W$. The expression $+/X \times W$ is commonly called an "inner product."

An important special case of weighed FNRS are those in which the weight vector, W, is obtained from a so-called radix vector,

$$B = B[1], B[1], B[2], . . . , B[N].$$

Usually B[I] is an element of Z, the set of integers (negative radices are also extensively discussed in the literature). The prospect of B[I] being a complex number has also been proposed.

Definition 7 A FNRS is a weighed radix system if it is a weighed system (Def. 6) in which the elements of the weight vector W are defined as follows:

$$W[N] = 1,$$

$$W[I] = W[I+1] \times B[I+1] \text{ for } I = N-1, N-2, \ldots, 1.$$

where $B[I] \in Z$ is an element of a radix vector.

Note that in our choice of the definition for W we are continuing to restrict ourselves to the representation of integers.

Definition 8 A weighed FNRS is a fixed radix (or base) system if all elements of B are the same, i.e., if $B[I] = B[J]$ for $1 \leq I,J \leq N$.

Definition 9 A weighed FNRS is a mixed radix (or base) system if all elements of B are not the same, i.e., if there exits some $I \neq J$ $(1 \leq I,J \leq N)$ such that $B[I] \neq B[J]$.

Fixed radix number systems are the most commonly used. In this case, F, the function between symbols and interpretation, may be specified as a polynomial ("radix polynomial") with the digit vector comprising the coefficients. A very interesting treatment of fixed radix FNRS, based upon the ring of polynomials over the integers may be found in [12]. Radix polynomial FNRS are also called "polyadic" FNRS.

The computer language APL includes a primitive operator which mechanizes the mapping from digit vectors to integers for the radix number systems. If we represent the elements of the set of representable numbers ("the interpretation set") using standard sign and magnitude decimal notation, then

$$Q \leftarrow B \perp X$$

produces an element $Q \in RE$ where B is a radix vector and X is a digit vector. This APL operator is called DECODE. For example if

$$B \leftarrow 2,2,2,2$$
$$X \leftarrow 1,1,1,0$$

then $Q \leftarrow B \perp X$ is the decimal equivalent of the binary digit vector 1, 1, 1, 0. If B is 30,24,60,60 and X is 5,7,15,37 then $Q \leftarrow B \perp X$ is the number of seconds in 5 days, 7 hours, 15 minutes, and 37 seconds.

Let us return now to a discussion of digit sets, i.e., the sets from which digit vector elements are taken. Recall that A, the set of digit vectors, was defined by

$$A = D1 \times D2 \times \ldots \times DN$$

where DI (I = 1,2,...,N) are sets of digits.

Definition 10 The canonical (or standard) digit set for the Ith element of a digit vector in a radix FNRS is the set

$$DI = \{0, 1, 2, ..., (B[I]-1)\}$$

where B[I] is the Ith element of the radix vector. Note that in this case $|DI| = B[I]$.

We will make use of other than canonical digit sets, for example, symmetric digit sets, defined as follows:

Definition 11 A symmetric digit set with respect to the positive integer K is the set
$ZoK = \{-K, -(K-1), \ldots 0, 1, \ldots, (K-1), K\}$.

Definition 12 A fixed radix FNRS with $B[I] \geq 2$ for all $1 \leq I \leq N$ with canonical digit sets is called a conventional FNRS.

EXAMPLES OF COMMONLY USED FINITE NUMBER REPRESENTATION SYSTEMS

The above definitions provide a framework for describing a large number of FNRS including residue, negative radix and signed-digit. Within the constraints of this paper, however, we will only review the definition of the generalized, positive radix versions of four commonly used FNRS: 1) Conventional, radix B, unsigned (magnitude only); 2) Conventional, radix B, sign and magnitude; 3) Conventional, radix B, radix complement; and 4) Conventional, radix B, diminished radix complement.

The definition of these FNRS are presented in Figures 2-5 using symbols defined in Figure 1. Each figure describes the symbol set, the interpretation set, and the symbol to interpretation (SI) function in the form of an APL function. These APL functions, generalizations of the APL decode ($\perp$) operator, have digit-vectors as arguments and produce a sign and magnitude, decimal representation of the corresponding element of the interpretation set. We'll use familiar notation to denote a particular element of the interpretation set.

PRIMITIVE OPERATIONS ON COMMON FINITE NUMBER REPRESENTATION SYSTEMS (DIGIT VECTOR ALGORITHMS)

To this point we have concentrated on the representation of finite sets of numbers using symbol sets consisting of digit vectors. We now present fundamental unary and binary operations on symbol sets which, together with isomorphic SI mappings such as described in the previous section, will enable us to simulate useful algebraic structures. Examples of the operations or "digit vector algorithms" include sum, difference, inverse, range extension, and range contraction. Note that in defining these algorithms we are assuming that we are given standard integer arithmetic on the individual digits.

In the remainder of this paper we describe primitive digit vector algorithms which the designer of a multi-valued logic processor must be prepared to implement. The proposed set is motivated by Avizienis [13]. Space does not permit a discussion of each algorithm, however, as an example of what might be done, we include a discussion of the SUM and CARRY DVAs.

SUM and CARRY Digit Vector Algorithms

The digit vector algorithm, SUM, corresponds to what we commonly call addition. Given an SI mapping F, we need to define SUM such that

$$F(X \text{ SUM } Y) = F(X) +M\ F(Y)$$

where $X,Y \in AE$.

For number systems (fixed or mixed base) with all elements of the radix vector, $B \geq 2$, and canonical digit set, the following digit vector operation applies:

$$S \leftarrow B \mid X + Y + C$$

where

B is the radix vector;
X and Y are the digit vector operands with $\rho X = \rho Y$,
C is the carry vector (to be defined),
and S is the sum vector.

The sum digit for a given position of an adder, say the Ith position, is sometimes given as

$$S[I] \leftarrow (X[I] + Y[I] + C[I]) - B[I] \times C[I-1]$$

where C[I] is the carry into the position, C[I-1] is the carry out, and B[I] is the Ith element of the radix vector. This form may better correspond to intuition, namely, that the sum digit is the sum of the two operand digits and the carry in (X[I] + Y[I] +C[I]) minus a correction. If the sum is greater than the maximum digit we can represent (B[I]-1) then we subtract B[I] from the Ith position and add 1 in the position I-1. Since the weight of position I-1 is $W[I-1] = B[I] \times W[I]$, this subtraction of B[I] in position I and addition of 1 in position I-1 do not change the value represented by the digit vector.

Listing of DYAs for common FNRS

We conclude by offering an annotated listing of primitive digit vector algorithms for the four common finite number representation systems we have reviewed. In exchange for the reader's willingness to read APL, he/she will find a formal description of operations which should be implemented for a favorite choice of radix, B, and choice of common number system. Similar work on less conventional but potentially practical number systems is underway and the interested reader is invited to contact the author for further information.

REFERENCES

1. Sandra Pakin, APL 360 Reference Manual, Second Edition, SRA, Chicago, 1972.
2. G. Demars, J. C. Rault, G. Ruggio, Le Langage et Les Systems APL, (in French), Masson et Cie, Paris, 1974.
3. APL 360 User's Manual, IBM Data Processing Division.
4. APL Shared Variables (APLSV) User's Guide, IBM (SH20-1460).
5. APL Language, IBM, (GC26-3847).
6. University of Michigan Computing Center Memo 363, MTS APL User's Guide.
7. L. Gilman and A. Rose, APL 360, An Interactive Approach, John Wiley and Sons, New York, 1970.
8. K. E. Iverson, A Programming Language, Wiley, New York, 1962.
9. H. Katzan, APL Programming and Computer Techniques, van Nostrand Reinhold, 1970.
10. S. Budkowski, D. Atkins, "A unified classification of finite number systems," Systems Engineering Lab. Report No. 106, ECE Dept., University of Michigan, Ann Arbor.
11. D. Atkins, The role of redundancy in computer arithmetic," Computer, June 1975, Vol. 8, No. 6, pp. 74-76.
12. D. Matula, "Radix arithmetic: Digital algorithms for computer architecture," Chapter 9 in Applied Computation Theory: Analysis, Design, Modeling, R. Yeh, ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
13. A. Avizienis, "Digital Computer arithmetic: A unified algorithmic specification," Proceedings of the Symposium on Computers and Automata, New York, 1971, Polytechnic Press, New York. Available through Wiley-Interscience.


```
A = SET OF ALL DIGIT VECTORS.
AF = SET OF ALL LEGAL DIGIT VECTOR (THOSE FOR WHICH F IS DEFINED)
R = THE SET OF REAL NUMBERS.
RF = THE INTERPRETATION SET, I.E. THE IMAGE OF AF UNDER F.
B = FIXED VALUE OF ALL ELEMENTS OF THE RADIX VECTOR. B≥2.
ZB←⍳B = THE SET OF INTEGERS MODULO B.
X = ELEMENT OF AF, I.E. A LEGAL DIGIT VECTOR.
Q = ELEMENT OF RF.
⍴X = LENGTH OF DIGIT VECTOR, X.
B*N = B TO THE POWER N.
N CP X = N-ARY CARTESIAN PRODUCT OF SET X, I.E. X×X×...×X, N TIMES.
1↑X = THE FIRST ELEMENT OF VECTOR X.
1↓X = ALL BUT THE FIRST ELEMENT OF X.
⌊K = FLOOR OF K, I.E. THE GREATEST INTEGER ≤K.
⌈K = CEILING OF K, I.E. THE SMALLEST INTEGER ≥K.
```

FIGURE 1. SYMBOLS USED IN DEFINING NUMBER SYSTEMS IN FIGURES 2-5


```
SYMBOL SET: A←AF← N CP ⍳B

INTERPRETATION SET: RF←⍳(B*N)

SI FUNCTION: F:AF→RF

    ∇ Q←B DECODEUS X
[1]   ⍝SI FUNCTION FOR CONVENTIONAL RADIX B UNSIGNED FNRS
[2]   Q←B⊥X
    ∇
EXAMPLES

2 DECODEUS 0 1 1 0
6

2 DECODEUS 1 0 0 1
9

10 DECODEUS 0 3 1 9
319

5 DECODEUS 1 2 3 4
194

8 DECODEUS 3 5 7 7
1919
```

FIGURE 2. DEFINITION OF CONVENTIONAL, RADIX B UNSIGNED FNRS.

```
SYMBOL SET: A←AF← S × N CP ⍳B
     WHERE S←(0,1) AND N IS THE LENGTH OF THE DIGIT VECTOR
     REPRESENTING THE MAGNITUDE. IN S, 0 DENOTES + AND 1 DENOTES -
     NOTE THAT IF X∊AF THEN N=(⍴X)-1

INTERPRETATION SET: RF← -K,...,-1,0,1,2,...,K
     WHERE K←(B*N)-1

SI FUNCTION: F:AF→RF

    ∇ Q←B DECODESM X
[1]   ⍝SI FUNCTION FOR CONVENTIONAL RADIX B SIGN AND MAGNITUDE FNRS
[2]   Q←(¯1*1↑X)×B⊥1↓X
    ∇
EXAMPLES

2 DECODESM 0 1 1 0
6

2 DECODESM 1 0 0 1
¯1

10 DECODESM 0 3 1 9
319

5 DECODESM 1 2 3 4
¯69

8 DECODESM 0 5 7 7
383
```

FIGURE 3. DEFINITION OF CONVENTIONAL, RADIX B SIGN AND MAGNITUDE FNRS.

```
SYMBOL SET: A←AE←N CP ⍳B

INTERPRETATION SET:
     RADIX B, EVEN: RE←-K,...,-1,0,1,...,(K-1)
     WHERE K←(B*N)÷2

     RADIX B, ODD: RE← -K1,...,-1,0,1,...,K2
     WHERE K1←(⌊B÷2)×B*(N-1) AND K2←((B*(N-1))×(⌈B÷2))-1

SI FUNCTION: F:AE→RE

    ∇ Q←B DECODERC X
[1]   ⍝ SI FUNCTION FOR CONVENTIONAL RADIX B, RADIX COMPLEMENT FNRS
[2]   Q←(B⊥X)-((1↑X)≥B÷2)×B*⍴X
[3]   ⍝NOTE THAT FOR THE SPECIAL CASE OF B=2 THE FOLLOWING APPLIES:
[4]   ⍝Q←2⊥(-1↑X),1↓X
[5]   ⍝I.E. WE TREAT THE SIGN INDICATOR DIGIT AS HAVING NEGATIVE WEIGHT.
    ∇
EXAMPLES

2 DECODERC 1 0 0 1
¯7

10 DECODERC 0 3 1 9
319

5 DECODERC 3 2 3 4
¯181

8 DECODERC 3 5 7 7
1919
```

FIGURE 4. DEFINITION OF CONVENTIONAL, RADIX B, RADIX COMPLEMENT FNRS.

```
SYMBOL SET: A←AE←N CP ⍳B

INTERPRETATION SET:
     RADIX B, EVEN: RE←-K,...,-1,-0,+0,1,...,K
     WHERE K←((B*N)÷2)-1

     RADIX B, ODD: RE←-K1,...,-1,-0,+0,1,...,K2
     WHERE K1←((⌊B÷2)×B*(N-1))-1 AND K2←((B*(N-1))×⌈B÷2)-1

SI FUNCTION: F:AE→RE

    ∇ Q←B DECODEDRC X
[1]   ⍝SI FUNCTION FOR CONVENTIONAL RADIX B, DIMINISHED
[2]   ⍝ RADIX COMPLEMENT FNRS
[3]   Q←(B⊥X)-((1↑X)≥B÷2)×(B*⍴X)-1
    ∇
EXAMPLES

2 DECODEDRC 1 0 0 1
¯6

10 DECODEDRC 0 3 1 9
319

5 DECODEDRC 3 2 3 4
¯180

8 DECODEDRC 3 5 7 7
1919
```

FIGURE 5. DEFINITION OF CONVENTIONAL, RADIX B, DIMINISHED RADIX COMPLEMENT FNRS.

DIGIT VECTOR ALGORITHMS
COMMON FINITE NUMBER REPRESENTATION SYSTEMS
FIXED-RADIX
JANUARY 1978, REVISION 4

THESE DIGIT VECTOR ALGORITHMS ARE DEFINED FOR THE SPECIAL CASE OF FIXED-RADIX FNRS. A SCALAR GLOBAL VARIABLE B, WHICH MUST BE ≥ 2, IS THE RADIX.

ABBREVIATIONS FOR DIGIT VECTOR ALGORITHM FAMILY NAME:
SUM, CARRY
DIF (DIFFERENCE), BORROW
INV (ADDITIVE INVERSE)
SGN (SIGN DETECTION), EQZ (EQUAL ZERO)
REX (RANGE EXTENSION), RCN (RANGE CONTRACTION)
SDN (SCALE DOWN), SUP (SCALE UP)
PRD (PRODUCT)

ABBREVIATIONS FOR NUMBER SYSTEM NAME:
US - CONVENTIONAL, UNSIGNED
RC - RADIX COMPLEMENT
DRC - DIMINISHED RADIX COMPLEMENT
SM - CONVENTIONAL SIGN AND MAGNITUDE

ABBREVIATIONS FOR SINGULARITIES:
OVF - OVERFLOW
AN - ANOMALY
TR - TRUNCATIONS
THE ABOVE ARE FOLLOWED BY DVA FAMILY NAME AND NUMBER SYSTEM NAME, FOR EXAMPLE, OVFSUMUS.

ASSUMPTIONS:
1. FOR DYADIC FUNCTIONS ⍴X=⍴Y.
2. THE RADIX (B) IS ALWAYS A SCALAR. THE RADIX VECTOR FOR A DIGIT VECTOR X IS CONCEPTUALIZED TO BE (⍴X)⍴B. B MUST BE ≥ 2 BUT MAY BE ODD OR EVEN.
3. FOR ALL VARIATIONS OF SUM AND DIF, THE CARRY IN (CIN) AND BORROW IN (BIN) WILL BE DEFINED EXTERNAL TO THE APL FUNCTION.
4. IN RCN, SDN, AND SUP THE VALUE OF M MUST BE ≤ ⍴X FOR US, RC, AND DRC; M MUST BE < ⍴X FOR DRC.

```
    ∇ S←X SUMUS Y
[1]   ⍝SUM DVA CONVENTIONAL UNSIGNED FNRS.
[2]   ⍝ CIN AND A FIXED RADIX B ARE DEFINED EXTERNALLY
[3]   S←B|X+Y+X CARRYUS Y
[4]   OVFSUMUS←COUT
    ∇

    ∇ C←X CARRYUS Y;I;ORG
[1]   ⍝ CARRY DVA FOR CONVENTIONAL UNSIGNED FNRS.
[2]   ⍝ RADIX B AND CARRY IN CIN ARE EXTERNALLY DEFINED.
[3]   ORG←⍳1
[4]   I←(⍴X)+ORG-1
[5]   C←(((⍴X)-1)⍴0),CIN
[6]  LOOPCPR:→(ORG=I)/LCOUT
[7]   C[I-1]←B≤X[I]+Y[I]+C[I]
[8]   →(ORG<I←I-1)/LOOPCPR
[9]  LCOUT:COUT←B≤X[ORG]+Y[ORG]+C[ORG]
    ∇

    ∇ S←X SUMSM Y;SIGNX;SIGNY;SIGNS;MAGX;MAGY;MAGS
[1]   ⍝ SUM DVA FOR CONVENTIONAL SIGN AND MAGNITUDE FNRS.
[2]   SIGNX←SGNSM X
[3]   SIGNY←SGNSM Y
[4]   MAGX←1↓X
[5]   MAGY←1↓Y
[6]   →(SIGNX=SIGNY)/LEQSIGN
[7]   ⍝SIGNX ≠ SIGNY
[8]   BIN←0
[9]   MAGS←MAGX DIFUS MAGY
[10]  SIGNS←SIGNX≠BOUT
[11]  →(~BOUT)/LUEQSIGN
[12]  BIN←0
[13]  MAGS←MAGY DIFUS MAGX
[14] LUEQSIGN:S←SIGNS,MAGS
[15]  OVFSUMSM←0
[16]  →LZEROCHECK
[17]  CIN←0
[18] LEQSIGN:S←SIGNX,MAGX SUMUS MAGY
[19]  OVFSUMSM←COUT
[20]  ⍝FORCE ONLY + ZERO
[21] LZEROCHECK:→(0=EQZSM S)/0
[22]  S[⍳1]←0
    ∇

    ∇ S←X SUMRC Y
[1]   ⍝SUM DVA FOR RADIX COMPLEMENT FNRS.
[2]   S←X SUMUS Y
[3]   OVFSUMRC1←((SGNRC X)=SGNRC Y)∧(SGNRC X)≠SGNRC S
[4]   OVFSUMRC2←((SGNRC X)=SGNRC Y)∧OVFSUMUS∧~SGNRC X
[5]   OVFSUMRC←OVFSUMRC1∨OVFSUMRC2
    ∇

    ∇ S←X SUMDRC Y
[1]   ⍝ SUM DVA FOR DIMINISHED RADIX COMPLEMENT FNRS.
[2]   S←X SUMUS Y
[3]   CIN←COUT
[4]   S←X SUMUS Y
[5]   OVFSUMDRC1←((SGNDRC X)=SGNDRC Y)∧(SGNDRC X)≠SGNDRC S
[6]   OVFSUMDRC2←((SGNDRC X)=SGNDRC Y)∧OVFSUMUS∧~SGNDRC X
[7]   OVFSUMDRC←OVFSUMDRC1∨OVFSUMDRC2
    ∇
```



```
    ∇ D←X DIFUS Y
[1]   ⍝DIFFERENCE DVA FOR CONVENTIONAL,UNSIGNED FNRS.
[2]   D←B|(X-Y)-X BORROWUS Y
[3]   OVFDIFUS←BOUT
    ∇

    ∇ BRW←X BORROWUS Y;I;ORG
[1]   ⍝ BORROW DVA FOR CONVENTIONAL UNSIGNED FNRS
[2]   ORG←⍳1
[3]   I←(⍴X)+ORG-1
[4]   BRW←(((⍴X)-1)⍴0),BIN
[5]  LOOPBPR:→(ORG=I)/LBOUT
[6]   BRW[I-1]←((X[I]-Y[I])-BRW[I])<0
[7]   →(ORG<I←I-1)/LOOPBPR
[8]  LBOUT:BOUT←((X[ORG]-Y[ORG])-BRW[ORG])<0
    ∇

    ∇ D←X DIFSM Y;SIGNX;SIGNY;SIGND;MAGX;MAGY;MAGD
[1]   ⍝ DIFFERENCE DVA FOR CONVENTIONAL SIGNED MAGNITUDE FNRS.
[2]   SIGNX←SGNSM X
[3]   SIGNY←SGNSM Y
[4]   MAGX←1↓X
[5]   MAGY←1↓Y
[6]   →(SIGNX=SIGNY)/LUNEQSIGN
[7]   ⍝ *** SIGN(X) = SIGN(Y). ***
[8]   BIN←0
[9]   MAGD←MAGX DIFUS MAGY
[10]  SIGND←SIGNX≠BOUT
[11]  →(~BOUT)/LCOMBINE
[12]  BIN←0
[13]  MAGD←MAGY DIFUS MAGX
[14] LCOMBINE:D←SIGND,MAGD
[15]  OVFDIFSM←0
[16]  →LEQZCHECK
[17] LUNEQSIGN:CIN←0
[18]  D←SIGNX,MAGX SUMUS MAGY
[19]  OVFDIFSM←COUT
[20]  ⍝ *** FORCE ONLY POSITIVE ZERO ***
[21] LEQZCHECK:→(0=EQZSM D)/0
[22]  D[⍳1]←0
    ∇

    ∇ D←X DIFRC Y
[1]   ⍝ DIFFERENCE DVA FOR RADIX COMPLEMENT FNRS.
[2]   D←X DIFUS Y
[3]   OVFDIFRC1←((SGNRC X)=SGNRC Y)∧(SGNRC X)≠SGNRC D
[4]   OVFDIFRC2←((SGNRC X)=SGNRC Y)∧OVFDIFUS∧~SGNRC X
[5]   OVFDIFRC←OVFDIFRC1∨OVFDIFRC2
    ∇

    ∇ D←X DIFDRC Y
[1]   ⍝ DIFFERENCE DVA FOR DIMINISHED RADIX COMPLEMENT FNRS.
[2]   D←X DIFUS Y
[3]   BIN←BOUT
[4]   D←X DIFUS Y
[5]   OVFDIFDRC1←((SGNDRC X)=SGNDRC Y)∧(SGNDRC X)≠SGNDRC D
[6]   OVFDIFDRC2←((SGNDRC X)=SGNDRC Y)∧OVFDIFUS∧~SGNDRC X
[7]   OVFDIFDRC←OVFDIFDRC1∨OVFDIFDRC2
    ∇
```

```
    ∇ Y←INVSM X
[1]   ⍝ ADDITIVE INVERSE DVA FOR CONVENTIONAL
[2]   ⍝ SIGN AND MAGNITUDE FNRS
[3]   Y←(~1↑X),1↓X
[4]   ⍝ GENERATE AN ANOMALY SIGNAL TO INDICATE
[5]   ⍝ THAT A NEGATIVE ZERO IS GENERATED
[6]   ANINVSM←0=+/X
    ∇

    ∇ Y←INVRC X
[1]   ⍝ ADDITIVE INVERSE DVA FOR
[2]   ⍝ RADIX COMPLEMENT FNRS
[3]   CIN←1
[4]   Y←((⍴X)⍴0)SUMRC((⍴X)⍴B-1)-X
[5]   ⍝ OVERFLOW WILL OCCUR IF SIGN(X) = SIGN(Y)
[6]   ⍝ AND Y ≠ 0.
[7]   OVFINVRC←((SGNRC X)=SGNRC Y)∧~EQZRC Y
    ∇

    ∇ Y←INVDRC X
[1]   ⍝ ADDITIVE INVERSE DVA FOR
[2]   ⍝ DIMINISHED RADIX COMPLMENT FNRS
[3]   Y←((⍴X)⍴B-1)-X
[4]   ⍝ GENERATE AN ANOMALY SIGNAL TO INDICATE
[5]   ⍝ THAT A NEGATIVE ZERO IS GENERATED
[6]   ANINVDRC←0=+/X
[7]   ⍝ OVERFLOW WILL OCCUR IF SIGN(X) = SIGN(Y).
[8]   OVFINVDRC←(SGNDRC X)=SGNDRC Y
    ∇

    ∇ SIGN←SGNSM X
[1]   ⍝ SIGN TEST FOR SIGN AND MAGNITUDE
[2]   SIGN←1↑X
    ∇

    ∇ SIGN←SGNRC X
[1]   ⍝SIGN OF X IN RADIX COMPLEMENT FNRS.
[2]   ⍝ SIGN=0 IF X IS POSITIVE.
[3]   ⍝ SIGN=1 IF X IS NEGATIVE.
[4]   SIGN←(1↑X)≥B÷2
    ∇

    ∇ SIGN←SGNDRC X
[1]   ⍝SIGN OF X IN DIMINISHED RADIX COMPLEMENT FNRS.
[2]   ⍝ SIGN=0 IF X IS POSITIVE.
[3]   ⍝ SIGN=1 IF X IS NEGATIVE.
[4]   SIGN←(1↑X)≥B÷2
    ∇

    ∇ Z←EQZUS X
[1]   ⍝TEST OF ZERO FOR UNSIGNED FNRS
[2]   Z←∧/X=0
    ∇

    ∇ TEST←EQZSM X
[1]   ⍝ EQUAL ZERO TEST FOR SIGN AND MAGNITUDE FNRS
[2]   TEST←∧/1↓X=0
    ∇
```



```
    ∇ TEST←EQZRC X
[1]   ⍝ EQUAL ZERO TEST FOR RADIX COMPLEMENT FNRS
[2]   TEST←∧/X=0
    ∇

    ∇ TEST←EQZDRC X
[1]   ⍝ EQUAL ZERO TEST FOR DIMINISHED RADIX COMPLEMENT FNRS
[2]   TEST←(∧/X=0)∨(∧/X=B-1)
    ∇

    ∇ Z←M REXUS X
[1]   ⍝ CONVENTIONAL UNSIGNED FNRS RANGE EXTENSION
[2]   ⍝ X = DV TO BE EXTENDED
[3]   ⍝ M = NUMBER OF POSITIONS TO EXTEND (SCALAR)
[4]   Z←(M⍴0),X
    ∇

    ∇ Z←M REXSM X
[1]   ⍝ CONVENTIONAL SIGNED MAGNITUDE FNRS RANGE EXTENSION
[2]   ⍝ OPERANDS ARE DEFINED AS IN REXUS
[3]   Z←(1↑X),(M⍴0),1↓X
    ∇

    ∇ Z←M REXRC X
[1]   ⍝ RADIX COMPLEMENT FNRS RANGE EXTENSION
[2]   ⍝ OPERANDS ARE DEFINED AS IN REXUS
[3]   Z←((M⍴B-1)×SGNRC X),X
    ∇

    ∇ Z←M REXDRC X
[1]   ⍝ DIMINISHED RADIX COMPLEMENT FNRS RANGE EXTENSION
[2]   ⍝ OPERANDS ARE DEFINED AS IN REXUS
[3]   Z←M REXRC X
    ∇

    ∇ Z←M RCNUS X
[1]   ⍝ CONVENTIONAL UNSIGNED FNRS RANGE CONTRACTION
[2]   ⍝ X = DV TO BE CONTRACTED
[3]   ⍝ M = NUMBER OF POSITIONS TO CONTRACT (SCALAR)
[4]   Z←M↓X
[5]   OVFRCNUS←1≠∧/0=M↑X
    ∇

    ∇ Z←M RCNSM X
[1]   ⍝ CONVENTIONAL SIGN AND MAGNITUDE FNRS RANGE CONTRACTION
[2]   ⍝ OPERANDS ARE DEFINED AS IN RCNUS
[3]   Z←(1↑X),M↓1↓X
[4]   OVFRCNSM←1≠∧/0=M↑1↓X
    ∇

    ∇ Z←M RCNRC X
[1]   ⍝ RADIX COMPLEMENT FNRS RANGE CONTRACTION
[2]   ⍝ OPERANDS ARE DEFINED AS IN RCNUS
[3]   Z←M↓X
[4]   OVFRCNRC←((SGNRC Z)∧1≠∧/(B-1)=M↑X)∨(~SGNRC Z)∧~EQZRC M↑X
    ∇
```

```
    ∇ Z←M RCNDRC X
[1]   ⍝ DIMINISHED RADIX COMPLEMENT FNRS RANGE CONTRACTION
[2]   ⍝ OPERANDS ARE DEFINED AS IN RCNUS
[3]   Z←M RCNRC X
[4]   OVFRCNDRC←OVFRCNRC
    ∇
```

```
    ∇ Z←M SDNUS X
[1]   ⍝ CONVENTIONAL UNSIGNED FNRS SCALE DOWN
[2]   ⍝ X = DV TO BE SCALED DOWN
[3]   ⍝ M = NUMBER OF PLACES TO SCALE
[4]   Z←(Mρ0),(-M)↓X
[5]   TRSDNUS←1≠∧/0=(-M)↑X
    ∇
```

```
    ∇ Z←M SDNSM X
[1]   ⍝ CONVENTIONAL SIGNED MAGNITUDE FNRS SCALE DOWN
[2]   ⍝ OPERANDS ARE DEFINED AS IN SDNUS
[3]   Z←(1↑X),(Mρ0),(-M)↓1↓X
[4]   TRSDNSM←1≠∧/0=(-M)↑1↓X
    ∇
```

```
    ∇ Z←M SDNRC X
[1]   ⍝ RADIX COMPLEMENT FNRS SCALE DOWN
[2]   ⍝ OPERANDS ARE DEFINED AS IN SDNUS
[3]   Z←(Mρ(B-1)×SGNRC X),(-M)↓X
[4]   TRSDNRC←1≠∧/0=(-M)↑X
    ∇
```

```
    ∇ Z←M SDNDRC X;SIGN
[1]   ⍝ DIMINISHED RADIX COMPLEMENT FNRS SCALE DOWN
[2]   ⍝ OPERANDS ARE DEFINED AS IN SDNUS
[3]   Z←(MρSIGN←(B-1)×SGNDRC X),(-M)↓X
[4]   TRSDNDRC←1≠∧/SIGN=(-M)↑X
    ∇
```

```
    ∇ Z←M SUPUS X
[1]   ⍝ CONVENTIONAL UNSIGNED FNRS SCALE UP
[2]   ⍝ X = DV TO BE SCALED
[3]   ⍝ M = NUMBER OF PLACES TO SCALE
[4]   ⍝M<ρX
[5]   Z←(M↓X),Mρ0
[6]   OVFSUPUS←1≠∧/0=M↑X
    ∇
```

```
    ∇ Z←M SUPSM X
[1]   ⍝ CONVENTIONAL SIGN AND MAGNITUDE FNRS SCALE UP
[2]   ⍝X = DV TO BE SCALED. M= NUMBER OF PLACES TO BE SCALED.
[3]   ⍝M<ρX
[4]   Z←(1↑X),((M+1)↓X),Mρ0
[5]   OVFSUPSM←1≠∧/0=M↑1↓X
    ∇
```

```
    ∇ Z←M SUPRC X
[1]   ⍝ RADIX COMPLEMENT FNRS SCALE UP
[2]   ⍝ X = DV TO BE SCALED
[3]   ⍝ M = NUMBER OF PLACES TO SCALE
[4]   Z←(M↓X),Mρ0
[5]   OVFSUPRC←((SGNRC Z)∧1≠∧/(B-1)=M↑X)∨(~SGNRC Z)∧~EQZRC M↑X
    ∇
```



```
    ∇ Z←M SUPDRC X
[1]   ⍝ DIMINISHED RADIX COMPLEMENT FNRS SCALE UP
[2]   ⍝ X = DV TO BE SCALED
[3]   ⍝ M = NUMBER OF PLACES TO SCALE
[4]   Z←(M↓X),Mρ(B-1)×SGNDRC X
[5]   OVFSUPDRC←((SGNDRC Z)∧1≠∧/(B-1)=M↑X)∨(~SGNDRC Z)∧1≠∧/0=M↑X
    ∇
```

```
    ∇ P←M PRDUS X;PS;PC
[1]   ⍝ POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
[2]   ⍝ CONVENTIONAL, RADIX B, UNSIGNED FNRS
[3]   PS←B|M×X
[4]   PC←⌊(M×X)÷B
[5]   CIN←0
[6]   P←(1 REXUS PS)SUMUS 1 SUPUS 1 REXUS PC
    ∇
```

```
    ∇ P←M PRDSM X
[1]   ⍝ POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
[2]   ⍝ CONVENTIONAL, RADIX B, SIGN AND MAGNITUDE FNRS
[3]   P←(SGNSM X),M PRDUS 1 RCNUS X
    ∇
```

```
    ∇ P←M PRDRC X
[1]   ⍝ POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
[2]   ⍝ CONVENTIONAL, RADIX B, RADIX COMPLEMENT
[3]   P←M PRDUS X
[4]   →(0=SGNRC X)/0
[5]   ⍝ CORRECTION FOR NEGATIVE NUMBERS
[6]   CIN←0
[7]   P←P SUMUS(ρX)SUPUS(ρX)REXUS B-M
    ∇
```

```
    ∇ P←M PRDDRC X
[1]   ⍝ POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
[2]   ⍝ CONVENTIONAL, RADIX B, DIMINISHED RADIX COMPLEMENT FNRS
[3]   P←M PRDRC X
[4]   →(0=SGNDRC X)/0
[5]   ⍝ CORRECTION FOR NEGATIVE NUMBERS
[6]   CIN←0
[7]   P←P SUMUS(ρX)REXUS M-1
    ∇
```